ORNL/TM-13254

ornl

OAK RIDGE NATIONAL LABORATORY

LOCKHEED MARTIN

Performance Modeling for SPMD Message-Passing Programs

Jürgen Brehm
Patrick H. Worley
Manish Madhukar

MANAGED AND OPERATED BY
LOCKHEED MARTIN ENERGY RESEARCH CORPORATION
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

ORNL/TM-13254

405

Computer Science and Mathematics Division

Mathematical Sciences Section

PERFORMANCE MODELING FOR SPMD MESSAGE-PASSING PROGRAMS

Jürgen Brehm †
Patrick H. Worley •
Manish Madhukar ‡

† University of Hannover, Institut für Rechnerstrukturen und Betriebssysteme, Lange Laube 3, 30159 Hannover, Germany
• Oak Ridge National Laboratory, Mathematical Sciences Section, P. O. Box 2008, Oak Ridge, TN 37831-6367
‡ Computer Science Department, Vanderbilt University, Box 1679, Station B, Nashville, TN 37235

Date Published: June, 1996

Research was supported by the Mathematical, Information and Computational Sciences Division of the Office of Computational and Technology Research Program, Office of Energy Research, U.S. Department of Energy

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
managed by
Lockheed Martin Energy Research Corp.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC05-96OR22464

Contents

PERFORMANCE MODELING FOR SPMD MESSAGE-PASSING PROGRAMS

Jürgen Brehm
Patrick H. Worley
Manish Madhukar

Abstract

Today's massively parallel machines are typically message-passing systems consisting of hundreds or thousands of processors. Implementing parallel applications efficiently in this environment is a challenging task, and poor parallel design decisions can be expensive to correct. Tools and techniques that allow the fast and accurate evaluation of different parallelization strategies would significantly improve the productivity of application developers and increase throughput on parallel architectures.

This paper investigates one of the major issues in building tools to compare parallelization strategies: determining what type of performance models of the application code and of the computer system are sufficient for a fast and accurate comparison of different strategies. The paper is built around a case study employing the Performance Prediction Tool (PerPreT) to predict performance of the Parallel Spectral Transform Shallow Water Model code (PSTSWM) on the Intel Paragon.

PSTSWM is a parallel application code that was designed to evaluate different parallel strategies for the spectral transform method as it is used in climate modeling and weather forecasting. Multiple parallel algorithms and algorithm variants are embedded in the code. PerPreT uses a relatively simple algebraic model to predict execution time for SPMD (Single Program Multiple Data) parallel applications. Applications are modeled through parameterized formulae for communication and computation, where the parameters include the problem size, the number of processors used to execute the program, and system characteristics (e.g., setup times for communication, link bandwidth, and sustained computing performance per processor).

In this paper we describe performance models that predict the performance of the different algorithms in PSTSWM accurately enough to allow them to be compared, establishing the feasibility of such a demanding application of performance modeling. We also discuss issues in generating and validating the performance models, emphasizing the practical importance of tools such as PerPreT in such studies.

1. Introduction

Advances in microprocessor technology and interconnection networks have made it possible to construct parallel systems with a large number of processors (e.g., Cray Research T3D, IBM SP2, Intel Paragon, workstation networks running PVM). Unfortunately, the application programs developed for conventional sequential systems or for pipelined supercomputers do not automatically run efficiently on these systems. There are few tools to support the development of parallel programs, and the performance of parallel programs is strongly dependent on the parallel programming skills of the application developer.

Before writing a program, the developer must identify a parallelization strategy. In most cases there are many options for distributing the data and tasks onto the processors. These options often have widely varying performance characteristics that are functions of numerous system and program parameters, and it can be difficult to predict *a priori* which options are best. Accurate prediction of the performance trade-offs of alternative strategies and of how the performance will change as program parameters change would greatly benefit program developers.

As an example, several parallelization strategies have been proposed for global atmospheric circulation models that use the spectral transform numerical technique [15]. These codes have strict performance requirements, being used for weather forecasts or for long term climate simulations, and even small improvements in performance can be significant. Researchers have demonstrated empirically the performance of one or two strategies [3], [7], [14], [18], [23], [26], or have made qualitative or asymptotic comparisons between strategies using simple performance models [5], [6], [13], but this work only establishes the feasibility of the different approaches. To accurately compare the different strategies, researchers at Argonne National Laboratory and Oak Ridge National Laboratory developed the Parallel Spectral Transform Shallow Water Model (PSTSWM). Multiple parallel algorithms and algorithm variants are embedded in PSTSWM, allowing good algorithms to be identified from empirical studies. The results of the studies using PSTSWM have been extremely useful; however, PSTSWM took over two years to develop and the experiments required to identify the best algorithms are time consuming. We hope that performance models would be simpler to adapt to proposed changes in the application codes and could be used to quickly examine the effect of running on new machines or with different problem or machine parameters.

Several approaches for the modeling of parallel systems have been presented that use Markov models or Petri nets [12], [21], [22]. Unfortunately, it is difficult to apply these approaches to massively parallel systems:

- The graphical representation required by these approaches is very complex for systems with hundreds or thousands of processors.

- The parallel application description required is very detailed.

- The resulting systems of equations defining the models are large and expensive to solve.

Applications for massively parallel systems typically use the single program multiple data (SPMD) programming model and are loosely synchronous [2]. For such programs, simpler modeling techniques utilizing algebraic abstractions of the application and computer system can often be used without a significant loss of accuracy [1]. These techniques make it feasible to model architectures with thousands of processors and the resulting models can be evaluated quickly.

Recent research utilizing algebraic performance models includes [4], [17], and [19]. These papers focus on tools or methodologies, many of them language or system specific, that automatically generate performance models from source code and user input. The paper by Sarukkai *et. al.* [19] on a methodology and toolkit for the scalability analysis of message-passing parallel programs has similarities with our research, but our concerns are somewhat different. We are primarily interested in investigating the accuracy of algebraic performance models. We want to identify what types of models can be used when modeling full application codes in the context of comparing parallelization strategies. In earlier work we found that the different phases of a parallel code place both implementation and performance constraints on each other, and that evaluation of kernels in isolation can be misleading, especially in a prototyping environment. We feel that it is still an open question as to how to model full application codes. How complex must a model be to be sufficiently accurate? How can a model be validated and the model accuracy determined? How does the accuracy of a model "scale" with the number of processors, problem size, and other program and system parameters? The comparison of parallelization strategies is also an interesting application of modeling. It is a strict test in that it requires multiple accurate models, but also requires only relative accuracy. The goal is "fairness" in the models for the different strategies.

In this paper we show that a reasonably accurate prediction of performance measures is possible without requiring detailed application and system characterizations. We describe a case study employing algebraic models to predict the performance of the Parallel Spectral Transform Shallow Water Model code (PSTSWM) on the Intel Paragon. We use these models to determine which parallel algorithm options are optimal for a given problem size and number of processors. We determine the error in our predictions empirically. We concentrate on the feasibility of such an approach for comparing parallelization strategies We do not address directly how to generate accurate models before the application code has been written, but the results do provide guidance on how accurate the models need to be.

This research was possible only because of the prior existence of a number of tools: PSTSWM, PICL, and PerPreT. PSTSWM is a convenient testbed for such studies. PICL (Portable Instru-

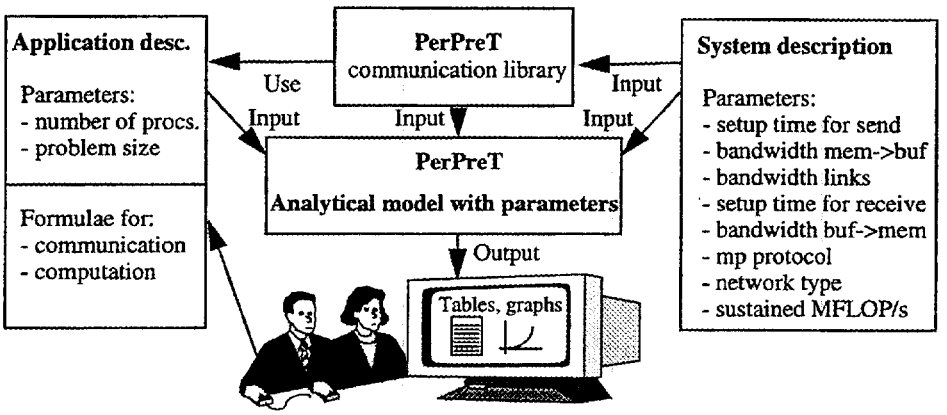


Figure 1: PerPreT Modules

mented Communication Library) was used to collect the performance data needed to construct and to validate the performance models [9], [27]. PerPreT (Performance Prediction Tool) was used to define and evaluate the performance models [1]. All three tools are available via the World Wide Web from the following locations:

- PerPreT: `http://www.irb.uni-hannover.de/~brehm/publications`
  PICL: `http://www.epm.ornl.gov/picl`
  PSTSWM: `http://www.epm.ornl.gov/chammp/pstswm`

The remainder of this paper is organized as follows. §2 is a description of how to use the performance prediction tool PerPreT. §3 is a brief description of the Intel Paragon. §4 is a description of the PSTSWM code and of the different parallelization strategies. §5 is a description of the parameterized PerPreT formulae for PSTSWM. §6 is a description of the modeling experiments and an analysis of the results. §7 is a discussion of our conclusions and some ideas for future work.

2. PerPreT

2.1. Overview

The high-level modules of PerPreT (i.e., application description, system description, communication library, analytical model) are outlined in Fig. 1. PerPreT uses parameterized system and application descriptions. Both the system and application descriptions are split into parameterized communication and computation descriptions. The system and application descriptions are kept independent of each other. Thus, applications are modeled on different systems without the need of defining new application descriptions.

An SPMD application is reduced to formulae for computation (number of arithmetic statements) and communication (calls to the communication library). The problem size for an application and the number of processors used to execute the SPMD program are free parameters. For modeling complex codes such as PSTSWM, PerPreT supports splitting the code into different computation phases according to their performance behavior. If extra operations for parallel computing are necessary (e.g., copy operations to prepare for communication), such extra phases can also be modeled with their performance characteristics.

PerPreT uses the system description parameters in Fig. 1 and a communication library to model the communication and computation behavior of the target architecture. The sustained MFlop/s (millions of floating point operations per second) rates and the rates used for extra phases (e.g., copy rates) are the only system variables that sometimes change with different applications or with different phases of a single application. More details on PerPreT can be found in [1].

2.2. Application description

In many massively parallel systems, each processor has direct access only to its own local memory. The communication between different processors is realized using message passing. Even on parallel architectures that directly support a global address space, message passing is a popular programming paradigm, both for portability and for efficiency. (Message passing is often efficient because it is a convenient "discipline" for dealing with the nonuniform access behavior inherent in any scalable memory system.)

Code for massively parallel systems is written primarily using the SPMD programming model. In this model the same code is loaded on all execution units to perform the same or similar tasks on different sets of data. Synchronization and communication for the tasks are done at the user level. At the system level, each processor executes its own code. Because of data dependencies, the various tasks of an SPMD program may have to communicate during execution. When using hundreds or thousands of processors, the parallel codes must be fairly regular and well structured to avoid load balancing problems and remain deadlock free. Often, the codes have alternating phases of communication and computation or, at least, distinct phases containing both communication and computation that are separated by logical synchronization points.

In Fig. 2, an example SPMD program is outlined as a task graph. The circles represent the computational tasks and the arrows represent communication between tasks. A computation phase does not last longer than TCP_i time units (i=1,2,..,7) and a communication phase does not last longer than TCM_j time units (j=1,2,..,6). The assumption is that TCP_i and TCM_j are the maximum times for all tasks at levels i and j, respectively. In Fig. 3, a possible mapping

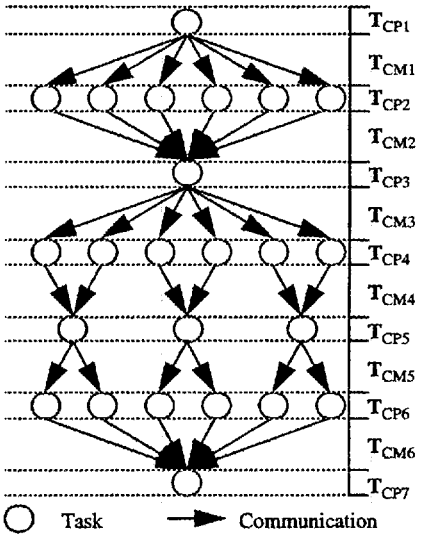


Figure 2: SPMD Program Task Graph

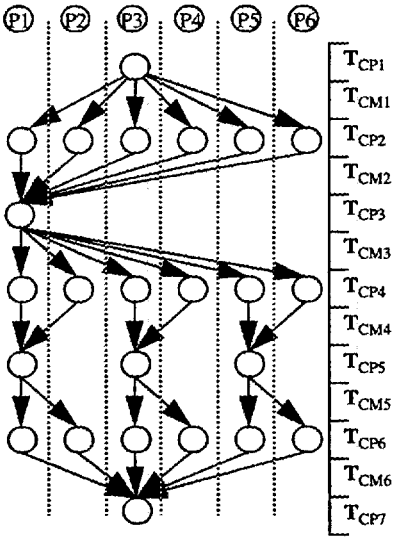


Figure 3: Mapping of an SPMD program on 6 processors

of the tasks onto processors (P1,..,P6) is shown. The estimated communication time of this mapping is:

$$\sum_j \mathrm{TCM}_j \tag{1}$$

The estimated computation time is:

$$\sum_i \mathrm{TCP}_i \tag{2}$$

The total estimated execution time is:

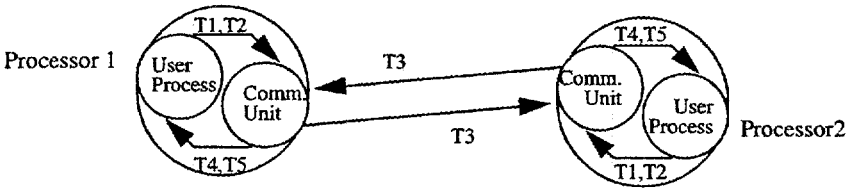
$$\sum_j \mathrm{TCM}_j + \sum_i \mathrm{TCP}_i \tag{3}$$

If there is tight synchronization between phases, the measured execution time will be very close to these estimates. However, many SPMD codes are only *loosely* synchronous, where synchronization between phases is enforced only by the natural data dependencies and by the explicit message passing used to satisfy these dependencies. For these codes, not all processors necessarily execute the same phase at the same time. If load imbalances at each phase are not all assigned to the same processors, then the use of maximum phase costs cause an overestimate of the total execution time. Such behavior can also be modeled in PerPreT, at the cost of more complexity in the models. In our experience and in the experiments described in this paper, the simple maximum phase cost model is sufficiently accurate, and is used exclusively in this paper.

For more general task graphs the number of subtasks per level, and thus the number of arrows per level, is not necessarily constant. Data parallelism often results in one subtask per processor for some of the levels, and the number of processors is a natural parameter in the communication and computation models. The problem size is the second parameter used. Clearly, the times TCP_i (determined by the number of statements to be executed) and TCM_j (determined by the message length) depend on these parameters, but the formulae for communication (1) and computation (2) are valid independent of the number of processors and the problem size.

2.3. System description

Communication. In most existing message-passing systems, the time required for each point-to-point communication request can be divided into the five phases outlined in Fig. 4. Depending on the message-passing protocol, one or more of the phases may or may not exist. For instance, transputers use synchronous message passing where the messages are copied directly from the user space on one processor to the user space on another processor. In this case it



T1: Send setup time. This time is needed for communication between the sender's communication unit and the sender's user process to initialize message buffers and to transfer control of the transmission to the communication unit.

T2: Send copy time. In the case of an asynchronous message-passing protocol, the outgoing message is often copied to a buffer controlled by the communication unit.

T3: Message transmission time. This time is required to copy the message from the sender's communication unit to the receiver's communication unit.

T4: Receive setup time. This time is needed for communication between the receiver's user process and the receiver's communication unit. The receiver's user process is informed about the location of the message.

T5: Receive copy time. In the case of an asynchronous message-passing protocol, the incoming message is often copied from a buffer controlled by the communication unit to the receiver's process space.

Figure 4: Message-Passing Communication

is not necessary to copy the messages from user space to the communication buffer and vice versa. The PerPreT approach is general enough to model a wide variety of existing message-passing protocols. The time for communication in a message-passing system normally follows the simple formula: Tc = T1+T2+T3+T4+T5 where Tc is the communication time. Some of the phases (e.g., T2, T3, T5) depend on the message size. If a complete system specification is available, these times can be used by the PerPreT communication library routines directly. However, users often do not have access to a detailed specification. The vendor provided times tend to be "optimistic", reporting best case times. These reported times may also be invalid if third party or other nonnative communication routines are used. For instance, if a program uses a portable communication library such as MPICH [10] or PICL [9], the times are slightly higher because of the overhead of an additional software layer. The PICL message-passing calls were used for this work, and the times T1,..,T5 were determined by experimentation [20]. These times are used as input parameters for the routines of the PerPreT communication library.

Computation. The computational behavior measured in MFlop/s of a single processor in a multiprocessor often shows a wide variation for different programs. Thus the performance of the processor for the given program has to be determined in order to predict the execution time of a program accurately. When the sequential or parallel code is available, PerPreT users preferably run the code on one or a small number of processors and calculate the sustained MFlop/s rate. If the code for an application is not available the PerPreT user has several choices:

- look at similar codes and take their performance characteristics;
- implement a small kernel to simulate the code;
- look at benchmarks that characterize the performance of the underlying hardware and system software.

In the case of PSTSWM the code was split into several compute and copy phases. The performance rate for each of these phases was determined from a set of runs of the program on eight processors, as described in §5.2. The assumption is that these values will prove to be accurate enough for experiments when more than eight processors are used. The validity of this assumption is examined in §6.

3. Intel Paragon

The Paragon XP/S MP system is a distributed memory multiprocessor in which the "nodes" are connected via a two-dimensional mesh interconnection network. Each node in the mesh

consists of three processors, two of which are dedicated to computation while the third is normally dedicated to communication. The communication processor is responsible for handling the messages generated by the node and the messages passing through the node. Processors and memory in a node are interconnected by a 400 MB/sec bus, and each link in the node interconnection network has a peak unidirectional bandwidth of 200 MB/sec.

There are three types of nodes - service, compute, and I/O. The service nodes host application control processes, compute nodes are assigned to parallel applications and dedicated to computations, and the I/O nodes provide the interface between the machine and RAID disks. The node interconnection network uses wormhole routing. The messages travel in the horizontal direction first and then in the vertical direction. Due to wormhole routing, communication latency is effectively distance independent.

The XP/S 150 MP at Oak Ridge National Laboratory consists of 1024 compute nodes in a 16 row by 64 column rectangular mesh. Each processor is a 50MHz i860XP, and all of the nodes have at least 64MB of "local" memory. In addition, there are 5 service nodes and 127 I/O nodes, each connected to a 4.8 GB RAID disk. At the time of these experiments, the system software was release 1.3 of OSF.

In our experiments, of the three processors in a compute node, one was used for computation, one was used for communication, and one was left idle. Using the second computation processor did not improve performance for PSTSWM due to the nature of the memory accesses. For the rest of the paper we will refer to a node in the XP/S 150 as a processor.

4. PSTSWM

PSTSWM is a message-passing parallel program that solves the nonlinear shallow water equations on a rotating sphere using the spectral transform method. PSTSWM is written in Fortran 77 with VMS extensions and a small number of C preprocessor directives. Message passing is implemented using MPI [16], PICL [9], PVM [8], or native message-passing libraries, with the choice being made at compile time. Optional performance instrumentation is implemented using the PICL trace and profile collection interface. PICL was used in the work described here, to collect performance data, but PICL simply represents a thin layer over the native NX message passing on the Intel Paragon.

The shallow water equations in the form solved by the spectral transform method describe the time evolution of three *state* variables: vorticity, divergence, and a perturbation from an average geopotential. The velocities are computed from these variables. PSTSWM advances the solution fields in a sequence of timesteps. During each timestep, the state variables of the problem are transformed between the physical domain, where the physical forces are calculated, and the spectral domain, where the terms of the differential equation are evaluated. The

1) Evaluate non-linear product and forcing terms.
2) Compute forward Fourier transform of non-linear terms.
3) Compute forward Legendre transforms.
4) Advance in time the spectral coefficients for the state variables.
5) Evaluate sums of spectral harmonics, simultaneously calculating the horizontal velocities from the updated state variables.
6) Compute inverse Fourier transform of state variables and velocities.

Figure 5: Outline of a single timestep of PSTSWM.

physical domain for a given vertical level is a tensor product longitude-latitude grid. The spectral domain for a given vertical level is the set of spectral coefficients in a truncated spherical harmonic expansion of the state variables.

Transforming from physical coordinates to spectral coordinates involves performing a real fast Fourier transform (FFT) for each line of constant latitude, followed by integration over latitude using Gaussian quadrature (approximating the Legendre transform (LT)) to obtain the spectral coefficients. The inverse transformation involves evaluating sums of spectral harmonics and inverse real FFTs. The basic outline of each timestep is described in Fig. 5. For more details on the steps in solving the shallow water equations using the spectral transform algorithm see [11].

The parallel algorithms in PSTSWM are based on decompositions of the physical and spectral computational domains over a logical two-dimensional processor mesh of size `PX` $\times$ `PY`. Initially, the longitude dimension of the physical domain is decomposed over the processor mesh row dimension and the latitude dimension is decomposed over the column dimension. Thus, FFTs in different processor rows are independent, and each row of `PX` processors collaborates in computing a "block" of FFTs. Similarly, the Legendre transforms in different processor columns are independent, and each column of `PY` processors collaborates in computing a "block" of Legendre transforms. The computation of the nonlinear terms at a given location on the physical grid is independent of that at other locations. The spectral domain decomposition is a function of the parallel algorithm used. In this version of PSTSWM, all computations on the spectral "grid" are likewise independent. Parallel efficiency is determined solely by the efficiency of the parallel algorithms used for the FFT and LT transforms and by any load imbalances caused by the choice of domain decomposition.

Two classes of parallel algorithms are available for each transform: distributed algorithms, using a fixed data decomposition and computing results where they are assigned, and transpose algorithms, remapping the domains to allow the transforms to be calculated sequentially. These represent four classes of parallel algorithms: distributed FFT/distributed LT, transpose

FFT/distributed LT, distributed FFT/transpose LT, and transpose FFT/transpose LT.

PSTSWM provides many parallel algorithms for each of the parallel algorithm classes [28]. In these experiments, we restrict ourselves to one transpose algorithm (for both FFT and LT), one distributed FFT algorithm, and two distributed LT algorithms, comprising the best parallel algorithms on the Intel Paragon. These algorithms are briefly described below.

Transpose. Assume that the transpose algorithm involves Q processors and that each processor contains D data to be transposed. Then every processor sends approximately D/Q data to every other processor, for a total of $\Theta(Q)$ messages and a total per processor volume of $\Theta(D)$.

Distributed FFT. Assume that the distributed FFT algorithm involves Q processors and that each processor contains D data to be transfomed. Then each processor exchanges $D/2$ data with its neighbors in a logical $(\log_2 Q)$-dimensional hypercube, for a total of $\Theta(\log Q)$ messages and a total per processor volume of $\Theta(D \log Q)$.

Distributed LT. Assume that the Legendre transform is parallelized over Q processors and that each processor will contain D spectral coefficients when the transform is complete. Then the per processor communication costs for the two distributed LT algorithms can be characterized by

- $\Theta(Q)$ messages, $\Theta(DQ)$ total volume
- $\Theta(\log Q)$ messages, $\Theta(DQ)$ total volume

respectively. The $\Theta(Q)$-step algorithm works on a logical ring, each processor communicating only with its two neighbors. The $\Theta(\log Q)$-step algorithm uses the same communication pattern as the distributed FFT algorithm.

These parallel algorithms for the FFT and LT generate the six parallel algorithms for the spectral transform method listed in Tab. 1. There are many implementation variants possible for

DH:	distributed FFT / $\Theta(\log Q)$-step distributed LT
DR:	distributed FFT / $\Theta(Q)$-step distributed LT
DT:	distributed FFT / transpose LT
TH:	transpose FFT / $\Theta(\log Q)$-step distributed LT
TR:	transpose FFT / $\Theta(Q)$-step distributed LT
TT:	transpose FFT / transpose LT

Table 1: Candidate PSTSWM parallel algorithms

each of these algorithms, distinguished, for example, by the choice of communication protocol

and the mapping of logical processors to physical processors. For these experiments, we use those implementations that have proven most efficient on the Intel Paragon. For details on the different implementation options, see [28].

PSTSWM is an interesting case study in modeling for many reasons. It has numerous distinct phases, each with its own computation and communication rates and patterns. It has (static) load imbalances that change with the choice of parallel algorithm and logical processor mesh. It requires significant global communication during each timestep, divided into two collective operations that access the processors in different ways. Finally, PSTSWM is a representative member of an important class of simulation models. In these studies, our goal is to build models that are accurate enough to indicate which parallel algorithm is most efficient for a given problem size and number of processors on a given multiprocessor.

5. Modeling PSTSWM

Assume that communication costs are negligible or scale linearly with the computation costs. Assume further that the computation rate varies in the same way across all algorithms as a function of the number of processors and of the problem size. Then a simple computational complexity analysis is sufficient to choose between the alternative parallel algorithms. If these assumptions do not hold or if runtime estimates are also needed, then we must determine both the computation and communication costs for a range of numbers of processors and of problem sizes.

In earlier research, we showed that different logical phases of a code may need to be modeled individually [25]. Each phase has its own computation rate, depending on the amount of computation and the amount and pattern of memory accesses. As the number of processors and problem size change, the percentage of time spent in each phase changes. This changes the overall computation rate. In the following, we identify and construct models for important phases. For brevity, we present only the phase models for algorithm TH. Models for the other parallel algorithms are given in Tab. 14-17 in the appendix.

5.1. Parameters

PerPreT expects one formula for the computation and one formula for the communication as input. These formulae use the number of processors and the problem size as parameters. For PSTSWM, the problem is specifed by 8 parameters: `DT`, `TAUE`, `MM`, `NN`, `KK`, `NLAT`, `NLON`, `NVER`, and by the specification of initial data and forcing function. The data and forcing function specification is fixed in these experiments and the following performance models are

specific to the particular test case[1], representing the calculation of solid body rotation steady state flow [24]. DT is the length of the timestep and TAUE is the duration of the model run in simulated time. Thus, TAUE/DT is the number of timesteps in the simulation. For these experiments the number of timesteps is fixed at 108. MM, NN, and KK determine which spectral coefficients are generated. We use the common choice of MM = NN = KK, which implies that MM + 1 Fourier coefficients are retained from the Fourier transform and (MM + 1)(MM + 2)/2 spectral coefficients are used in the spectral representation. NLAT, NLON, and NVER define the tensor-product physical grid of size NLON × NLAT × NVER. These values are also a function of MM when the computational complexity is minimized subject to satisfying an anti-aliasing condition. The number of processors used is specified by the logical processor mesh PX × PY.

The costs associated with each phase of PSTSWM are functions of the domain decomposition relevant to the phase. There are two decompositions of the physical domain (longitude × latitude × vertical levels):

- NLLON_P, NLLAT_P, and NLVER_P, denoting the number of local longitudes, latitudes, and vertical levels assigned to a given processor during physical domain computations,
- NLLON_F, NLLAT_F, and NLVER_F, denoting the number of local longitudes, latitudes, and vertical levels assigned to a given processor during the Fourier transform phases,

one decomposition of the Fourier domain (wavenumber × latitude × vertical levels):

- NLMM_S, NLLAT_S, and NLVER_S, denoting the number of local wavenumbers, latitudes, and vertical levels assigned to a given processor during the Legendre transform phases,

and one decomposition of the spectral domain (spectral coefficients × vertical levels):

- NLSP_S, NCSP_S, and NLVER_S, denoting the number of spectral coefficients assigned to a single processor and to a single column of processors, respectively, during computations in the spectral domain.

The values for these 11 parameters are functions of MM, NN, KK, NLAT, NLON, NVER, PX, PY, and the parallel algorithm being used. The values for parallel algorithm TH are as follows:

$\text{NLLON\_P} = \lceil \text{NLON/PX} \rceil$	$\text{NLLAT\_P} = 2 \cdot \lceil \text{NLAT}/(2 \cdot \text{PY}) \rceil$	$\text{NLVER\_P} = \text{NVER}$
$\text{NLLON\_F} = \text{NLON}$	$\text{NLLAT\_F} = 2 \cdot \lceil \text{NLAT}/(2 \cdot \text{PY}) \rceil$	$\text{NLVER\_F} = \lceil \text{NVER/PX} \rceil$
$\text{NLMM\_S} = \text{MM} + 1)$	$\text{NLLAT\_S} = 2 \cdot \lceil \text{NLAT}/(2 \cdot \text{PY}) \rceil$	$\text{NLVER\_S} = \lceil \text{NVER/PX} \rceil$
$\text{NCSP\_S} = (\text{MM} + 2)(\text{MM} + 1)/2$	$\text{NLSP\_S} = \text{NCSP\_S}$	

[1]Most of the other test cases differ only in calculation of the nonlinear terms, and only one phase model would need to be changed when changing cases.

The values for the other 5 algorithms are listed in Tab. 13 in the appendix. These are maximum values across all processes, and load imbalance enters via the floor and ceiling functions in the expressions. The load imbalance varies with logical grid aspect ratio and parallel algorithm, and between the different computational domains.

5.2. Computation model

PerPreT requires a simple algebraic expression for the number of arithmetic statements executed by each processor. If this number varies for different processors, the maximum is used. To implement different models for different phases, a separate algebraic expression is generated for each phase. The computation model for the entire program is a weighted sum of the phase expressions, where the weights are the computation rates associated with the different phases.

We include phases that involve only copying. In parallel codes, copying is often a significant cost. For example, for the transpose-based parallel algorithms the indices of the field arrays must be in a different order for the transposition than for the computation. This requires an explicit copy before and after the communication phases.

The following phase computation models for parallel algorithm TH were derived from the source code and are of two types: number of floating point computations and number of bytes copied. For the purposes of these experiments, we limited ourselves to (simple) models that an industrious application developer would be willing to generate. Some phases are interleaved in time even for a single timestep, and a given phase model represents the sum of all calls to the relevant code during one time step. Later we will examine whether this number of phases is necessary or sufficient.

The phase models come in two forms: one-parameter (single rate) and two-parameter models. All of the phases show some performance sensitivity to problem size and aspect ratio, but many of the computational phases are relatively insensitive and a single rate is sufficient. (We examine accuracy issues in detail in §6.) The variations in the rates in Tab. 2 between different phases arise from different access patterns to and from memory, and from differing amounts of computation per memory access.

In contrast, rates for phases with low computation to memory access ratios, like copy phases, vary significantly with aspect ratio and problem size. With a few exceptions, this variation is approximated reasonably well with the following two-parameter model: a rate for the total number of operations and a rate for the number of times that the inner loop is executed. The form of these models was derived empirically, but one justification is that it takes into account the additional cost of crossing cache and page boundaries when accessing memory.

The phases requiring two-parameter models and the rates for all models were determined empirically. Timings were taken from a series of 8-processor runs using two different problem

Phase	Model	Rate
		$(1/\mathbf{a}, 1/\mathbf{b})$
	physical domain computation	
1	$12 \cdot \texttt{NLLON\_P} \cdot \texttt{NLLAT\_P} \cdot \texttt{NLVER\_P}$	4.8
	forward FFT	
2	$\lceil(\texttt{PX}-1)/\texttt{PX}\rceil \cdot 32 \cdot \texttt{NLLAT\_P} \cdot \texttt{NLVER\_P} \cdot (\mathbf{a}+\mathbf{b} \cdot \texttt{NLLON\_P})$	(4.5, 23.1)
3	$\lceil(\texttt{PX}-1)/\texttt{PX}\rceil \cdot 32 \cdot \texttt{NLLAT\_P} \cdot \texttt{NLVER\_F} \cdot (\mathbf{a} \cdot \texttt{PX}+\mathbf{b} \cdot \texttt{NLLON\_F})$	(17.7, 21.6)
5	$20 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot \texttt{NLLON\_F} \cdot (\mathbf{a}+\mathbf{b} \cdot \log_2(\texttt{NLLON\_F}/4))$	(3.8, 24.0)
6	$64 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (\mathbf{a}+\mathbf{b} \cdot \texttt{NLLON\_F}/4)$	(4.0, 15.2)
7	$144 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (\mathbf{a}+\mathbf{b} \cdot \texttt{NLLON\_F}/4)$	(10.4, 19.8)
	forward LT	
9	$(\texttt{PY}-1) \cdot 6 \cdot \texttt{NLVER\_S} \cdot \texttt{NCSP\_S}/\texttt{PY}$	4.4
10	$61 \cdot \texttt{NLVER\_S} \cdot \texttt{NLMM\_S} \cdot \texttt{NLLAT\_S}$	10.0
11	$(14 \cdot \texttt{NLLAT\_S}-1) \cdot \texttt{NCSP\_S} \cdot \texttt{NLVER\_S}$	15.1
	spectral domain computation	
12	$13 \cdot \texttt{NLSC\_S} \cdot \texttt{NLVER\_S}$	11.5
	inverse LT	
13	$17 \cdot \texttt{NCSP\_S} \cdot \texttt{NLVER\_S}$	7.0
14	$(14 \cdot \texttt{NCSP\_S}+10 \cdot \texttt{NLMM\_S}) \cdot \texttt{NLLAT\_S} \cdot \texttt{NLVER\_S}$	12.8
17	$40 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (\mathbf{a}+\mathbf{b} \cdot (\texttt{NLLON\_F}/2-\texttt{NLMM\_S})$	(22.1, 36.8)
	inverse FFT	
18	$70 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (\mathbf{a}+\mathbf{b} \cdot \texttt{NLLON\_F}/4)$	(8.8, 20.4)
19	$40 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (\mathbf{a}+\mathbf{b} \cdot \texttt{NLLON\_F}/2)$	(2.8, 18.6)
20	$(25/2) \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot \texttt{NLLON\_F} \cdot (\mathbf{a}+\mathbf{b} \cdot \log_2(\texttt{NLLON\_F}/4))$	(3.8, 24.0)
21	$\lceil(\texttt{PX}-1)/\texttt{PX}\rceil \cdot 20 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot \texttt{NLLON\_F}$	10.2
22	$\lceil(\texttt{PX}-1)/\texttt{PX}\rceil \cdot 20 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLLON\_P} \cdot (\mathbf{a} \cdot \texttt{PX}+\mathbf{b} \cdot \texttt{NLVER\_P})$	(15.2, 18.6)

Table 2: Computational models and MFlop/s or MByte/s rates for algorithm TH

Phase	Description
	physical domain computations
1	nonlinear terms
	forward FFT
2	copy before transpose or copy before distributed computation
3	copy in transpose
4	distributed computation
5	sequential forward FFT
6	copy before communication for complex-to-real extraction
7	extract the real transform from the complex transform
	forward LT
8	copy before transpose
9	copy inside transpose or summation inside distributed vector sum
10	forward LT preprocessing
11	forward LT computation
	spectral domain computation
12	time update
	inverse LT
13	inverse LT preprocessing
14	inverse LT computation
15	copy before transpose
16	copy inside transpose
17	zero truncated coefficients
	inverse FFT
18	convert real transform data into complex transform data
19	copy after conversion
20	sequential inverse FFT
21	copy before transpose or copy before distributed computation
22	copy inside transpose
23	distributed computation

Table 3: Computation phase descriptions for all parallel algorithms.

sizes, 32 bit precision, and all possible aspect ratios (1x8, 2x4, 4x2, 8x1). For one-parameter models we use the maximum observed rates. This avoids contamination from atypical rates arising from inefficient memory alignments or poor cache performance. For the two-parameter models we use typical or median values, giving preference to rates for the smaller problem when there is a significant discrepancy. The intent is to better capture the behavior when extrapolating to larger numbers of processors. If a rate for a phase showed variation but could not be accurately fit with the type of two-parameter models described above, we use a one-parameter model.

Interactions with the memory hierarchy are major determiners of computation and copy rates, and these change in a phase as the problem and algorithm parameters vary. Even one-parameter phase models that are highly accurate for the 8-processor calibration runs will be valid only for a range of problem and machine parameters. Consequently, there will be errors in the rates when extrapolating, and scalability will be a problem even in a phase model approach. Algebraic models that take into account memory access patterns are possible, but such models are unlikely to be developed by an application programmer and are not discussed here. Our hope is that the range of validity of the rates is large enough or that the degradation affects all phases in a similar enough way that the algorithm comparisons will be reasonably accurate.

5.3. Communication model

PerPreT requires a high-level description of the communication in a parallel program. For PSTSWM, communication models are required for the two parallel FFT and for the three parallel LT algorithms. The detailed models are given in Tab. 4. The `comm(mess_length)` function in Tab. 4 returns the time needed for one communication between two processors of the multiprocessor. The parameter `mess_length` is the message length in bytes. Contention for bandwidth and other network resources and distance in the network are ignored in these experiments. The models are parameterized solely by the number of messages and by the size of each message for a given processor.

Note that the nature of the communication varies significantly between the different algorithms. The distributed FFT and $\Theta(\log Q)$-step distributed LT use a butterfly pattern in their communication. In the transpose algorithm, each processor sends to every other processor, using an exclusive-OR ordering to avoid some contention. In the $\Theta(Q)$-step distributed LT, each processor sends and receives from only two other processors, and the two processors are chosen to be neighbors in the physical network if possible. The $\Theta(Q)$-step distributed LT also attempts to overlap the communication with computation. None of these differences are taken into account in these models, although they could be, and this work also examines whether more detailed models are needed. More detailed models of the communication cost are known

Direction	Model
	Distributed FFT
forward	$\lceil (\mathtt{PX}-1)/\mathtt{PX} \rceil \cdot (1+\log_2(\mathtt{PX})) \cdot \mathtt{comm}(32 \cdot \mathtt{NLLAT\_P} \cdot \mathtt{NLVER\_P} \cdot \lceil \mathtt{NLLON\_P}/2 \rceil)$
inverse	$\lceil (\mathtt{PX}-1)/\mathtt{PX} \rceil \cdot (1+\log_2(\mathtt{PX})) \cdot \mathtt{comm}(20 \cdot \mathtt{NLLAT\_P} \cdot \mathtt{NLVER\_P} \cdot \lceil \mathtt{NLLON\_P}/2 \rceil)$
	Transpose FFT
forward	$(\mathtt{PX}-1) \cdot \mathtt{comm}(32 \cdot \mathtt{NLLAT\_P} \cdot \mathtt{NLVER\_F} \cdot \mathtt{NLLON\_P})$
inverse	$(\mathtt{PX}-1) \cdot \mathtt{comm}(20 \cdot \mathtt{NLLAT\_P} \cdot \mathtt{NLVER\_F} \cdot \mathtt{NLLON\_P})$
	$\Theta(Q)$-step distributed LT
forward	$(\mathtt{PY}-1) \cdot \mathtt{comm}(24 \cdot \mathtt{NLVER\_S} \cdot \mathtt{NLSP\_S})$
inverse	$(\mathtt{PY}-1) \cdot \mathtt{comm}(24 \cdot \mathtt{NLVER\_S} \cdot \mathtt{NLSP\_S})$
	$\Theta(\log Q)$-step distributed LT
forward	$\sum_{i=1}^{\log_2 \mathtt{PY}} 2 \cdot \mathtt{comm}\left(8 \cdot \lceil 3 \cdot \mathtt{NLVER\_S} \cdot \mathtt{NCSP\_S}/2^i \rceil\right)$
inverse	—
	Transpose LT
forward	$(\mathtt{PY}-1) \cdot \mathtt{comm}(64 \cdot \mathtt{NLLAT\_F} \cdot \mathtt{NLVER\_S} \cdot \mathtt{NLMM\_S})$
inverse	$(\mathtt{PY}-1) \cdot \mathtt{comm}(40 \cdot \mathtt{NLLAT\_F} \cdot \mathtt{NLVER\_S} \cdot \mathtt{NLMM\_S})$

Table 4: Communication models for forward and inverse transforms

to be necessary if poor communication algorithms or protocols are used. For example, a transpose algorithm in which all processors send to processor 0, then processor 1, etc., serializes the communication, and the maximum per processor number of messages and message volume will not represent the communication cost. The goal of the algorithm comparison is to compare good parallel implementations, and we hope that more detailed communication models are not necessary.

6. Experiments

The performance models described in the previous section and in the appendix are meant to be simple enough to be generated by the application developer, yet accurate enough to be used when scaling problem and machine parameters and when comparing alternative parallel algorithms. The approach taken here has been to construct the application model from a set of phase models.

In this section we begin by examining the accuracy of the individual phase models. We then use the models to investigate the following performance questions:

1) What is the best logical aspect ratio to use for a given parallel algorithm and for a given number of processors?

2) What is the best parallel algorithm to use for a given number of processors?

3) How long will the application take to complete a run?

Two problem sizes are investigated, denoted by T42 and T85,

	MM	NN	KK	NLAT	NLON	NVER
T42	42	42	42	64	128	16
T85	85	85	85	128	256	16

We discuss the phase model validation for algorithm TH and for three numbers of processors, $P = 8, 64, 128$. For the three performance questions, we discuss $P = 8, 16, 32, 64, 128, 256, 512$. The optimal logical aspect ratio is determined for each parallel algorithm. The optimal parallel algorithms are determined over all algorithms and aspect ratios. The estimation of runtimes is discussed in terms of the optimal parallel algorithms.

Finally, we reexamine the models, evaluating the effectiveness and importance of the phase model approach in being able to answer the stated performance questions.

6.1. Phase model validation

The rates for the phase models were determined from data for 8-processor experiments, as described earlier. Table 5 indicates the maximum error in using these simple one- or two-parameter models for a given phase over all possible aspect ratios, where the percentage absolute error is defined by

$$100 \cdot |\text{predicted\_time} - \text{true\_time}|/\text{true\_time} \; . \tag{4}$$

Observations on the accuracy of the phase models follow.

1) The maximum errors for the 8-processor runs used to determine the rates are small for the most part. There are some phases for which the simple one- and two-parameter models are not very accurate. These same phases also show poor accuracy for the 64- and 128-processor runs.

2) Many of the models are not very accurate when scaling to 64 and 128 processors *in the worst case*. What is not shown in this table is the range of validity across aspect ratio. Most of the models are quite accurate for all but the extreme aspect ratios. The dependence of the accuracy on aspect ratio can be inferred from Tab. 6, where the percentage error in the models is given for each aspect ratio in turn. The percentage error is defined to be

$$100 \cdot (\text{predicted\_time} - \text{true\_time})/\text{true\_time} \; . \tag{5}$$

Note that 256-processor results are included in Tab. 6 to provide additional information on the scalability of the models. Results for $P = 16, 32, 512$ are omitted because of space limitations.

	maximum percentage absolute error					
	T42			T85		
Phase	$P = 8$	$P = 64$	$P = 128$	$P = 8$	$P = 64$	$P = 128$
1	24.2	53.4	35.3	19.4	53.7	54.6
2	0.3	5.3	6.2	4.3	5.0	4.4
4	0.2	12.0	15.0	0.8	11.2	11.7
5	0.3	11.6	4.0	1.0	3.6	3.7
6	3.5	43.5	42.7	4.1	43.7	43.5
7	2.4	32.1	32.4	2.5	31.7	31.8
9	11.7	11.3	15.1	2.1	2.2	3.8
10	16.4	42.8	41.3	10.9	45.4	45.7
11	9.5	25.5	21.8	9.6	23.3	22.2
12	0.9	1.4	2.4	3.5	3.8	3.6
13	4.0	2.4	2.5	0.7	1.7	2.2
14	6.1	9.1	8.9	7.1	13.7	12.9
17	14.8	60.9	63.9	8.4	36.7	63.9
18	1.1	27.6	30.4	3.0	28.9	28.8
19	6.8	38.2	37.5	2.4	38.3	37.8
20	5.4	8.9	8.5	1.7	8.2	8.8
21	1.2	11.1	18.0	15.1	7.0	6.5
23	3.3	5.2	9.4	3.8	4.7	4.7
sum 1-23	3.3	6.1	8.5	2.3	10.6	9.5
FFT comm	40.8	16.0	36.0	13.5	13.3	29.0
LT comm	3.2	45.3	44.9	23.4	31.3	53.1

Table 5: Maximum percentage absolute error in phase models over all aspect ratios for algorithm TH.

Aspect Ratio PX × PY	T42 Runtime (seconds)	T42 % error in model	T85 Runtime (seconds)	T85 % error in model
8x1	92.08	-4.4	481.83	-4.3
4x2	90.84	-3.5	477.54	-3.5
2x4	90.67	-2.0	481.30	-3.2
1x8	83.61	-2.8	429.89	1.5
64x1	44.07	0.6	226.37	-0.9
32x2	22.98	-0.6	113.80	1.2
16x4	12.46	-1.7	60.21	1.2
8x8	12.80	-3.3	61.38	1.5
4x16	13.81	-3.1	66.18	0.6
2x32	16.75	-6.2	76.71	-1.0
1x64	—	—	95.23	-5.7
128x1	46.91	-0.8	227.18	-0.7
64x2	23.43	0.2	112.33	1.8
32x4	12.55	-2.3	58.31	1.8
16x8	7.12	-5.4	31.84	0.5
8x16	7.42	-5.5	34.20	-1.1
4x32	8.97	-9.9	39.92	-3.7
2x64	—	—	51.89	-7.3
256x1	—	—	240.45	-3.6
128x2	25.83	-1.1	116.85	-0.6
64x4	13.44	-2.8	59.67	-0.3
32x8	7.54	-7.5	32.51	-3.2
16x16	4.50	-11.4	19.19	-8.5
8x32	5.18	-16.3	21.90	-10.4
4x64	—	—	28.77	-15.2
2x128	—	—	—	—
1x256	—	—	—	—

Table 6: Runtime and model error for algorithm TH.

3) In general, the most inaccurate phase models are for those phases taking the least amount of time. This is to be expected given the greater degree of sensitivity to overhead and unpredictable memory access costs in short phases. This result is not directly observable from Tab. 5, but it can be inferred from the relatively good accuracy shown in the sum of the phase computation models ("sum 1-23") and in the total time predictions in Tab. 6.

4) Communication costs are not simple to separate from computation costs. The arrival of messages while a process is in a computation phase causes an overestimate of the computation cost and an underestimate of the communication cost. For completeness, we have included what data we have on communication costs and estimated the error in the models, but the accuracy of the communication models is best inferred from the accuracy of the total time predictions in Tab. 6.

5) While algorithm TH is not atypical, the accuracy of the models for the other algorithms varies from that shown here. The appendix contains results corrresponding to Tab. 6 for the other algorithms.

In summary, the individual phases are not always well modeled by using these simple performance models, but the phase model approach appears to be quite accurate when modeling the entire application.

6.2. Optimal aspect ratio

The first performance question of interest for PSTSWM is how to allocate processors among the different parallel transforms to minimize execution time, i.e., for a given number of processors, what logical aspect ratio should be used. The relative accuracy of the execution time predictions is important here, not the absolute accuracy. Table 7 describes the true and predicted optimum for different numbers of processors *when they differ*, and the percentage loss from using the model results. The loss is measured in the following way. Let PRED represent *the predicted* optimal aspect ratio. Let OPT represent the true optimal aspect ratio. The percentage loss is defined as

$$100 \cdot (\texttt{PRED}\_true\_time - \texttt{OPT}\_true\_time)/(\texttt{OPT}\_true\_time) \ . \tag{6}$$

Only 17 of the 84 model predictions are incorrect, and only 4 of these result in errors in runtime of more than 5%. Performance on the Paragon is very consistent, but there is some small variation between runs. The 7 cases in which the "error" is less than 1% should probably be considered correct.

What is not indicated in this table is how important it is to choose a good aspect ratio. The worst case aspect ratios are as much as ten times worse than the best case, primarily

Processors	T42 model results	T42 experimental results	T42 % error in runtime	T85 model results	T85 experimental results	T85 % error in runtime
DH (3 errors)						
32	4x8	2x16	0.5	4x8	4x8	—
512	16x32	32x16	8.5	16x32	32x16	0.2
DR (2 errors)						
32	1x32	4x8	1.2	1x32	1x32	—
512	16x32	32x16	16.8	32x16	32x16	—
DT (no errors)						
TH (2 errors)						
32	16x2	8x4	0.2	8x4	16x2	0.3
TR (4 errors)						
16	1x16	4x4	5.4	1x16	1x16	—
32	16x2	8x4	2.3	8x4	4x8	0.2
64	16x4	16x4	—	16x4	8x8	0.3
TT (6 errors)						
16	16x1	16x1	—	16x1	1x16	5.6
32	1x32	4x8	4.4	1x32	1x32	—
64	16x4	8x8	2.6	16x4	8x8	3.4
128	16x8	8x16	0.9	16x8	8x16	2.5

Table 7: Error in choosing optimal aspect ratio from model results instead of experimentally.

reflecting load imbalance. Tables 18-23 in the appendix contain more details on the sensitivity of performance to aspect ratio.

Determining a good logical aspect ratio is important when implementing a parallel strategy. A parallel code could incorporate the flexibility to change at least some of these parameters at compile-time or runtime, in which case PerPreT simply makes this more convenient to determine. This convenience should not be underestimated. Determining the optimal aspect ratio experimentally requires access to the same number of processors as will be used in a production run and numerous, possibly expensive, experiments.

6.3. Optimal parallel algorithm

Determining the optimal parallel algorithm experimentally requires developing, tuning, and evaluating multiple parallel implementations. This is much more time consuming than determining the optimal aspect ratio experimentally, and there is much to be gained from using performance models to predict the optimal parallel algorithm. As before, relative accuracy in the predicted execution times is what is important. Table 8 indicates the true and predicted optimal parallel algorithm for different numbers of processors, and the percentage loss from using the model-identified algorithm, measured as in (6). The optimal aspect ratio was found for each parallel algorithm before being compared with the other parallel algorithms. The

Processors	T42 model optimum	T42 experimental optimum	T42 % diff. in runtime	T85 model optimum	T85 experimental optimum	T85 % diff. in runtime
8	DR 1x8	DR 1x8	—	DT 1x8	DR 1x8	6.2
16	DT 1x16	DT 1x16	—	DT 1x16	DR 1x16	1.8
32	TR 8x4	TR 8x4	—	TR 16x2	TR 4x8	1.5
64	TR 16x4	TR 16x4	—	TR 16x4	TR 8x8	0.3
128	TH 16x8	TR 16x8	1.1	TT 16x8	TT 8x16	2.5
256	TH 16x16	TT 16x16	3.7	TT 16x16	TT 16x16	—
512	TH 16x32	TH 16x32	—	TT 16x32	TT 16x32	—

Table 8: Error in choosing optimal algorithm from model results instead of experimentally.

model results use the model-determined optimal aspect ratios. The empirical results use the experimentally-determined optimal aspect ratios.

The performance models correctly identify the optimal algorithm and aspect ratio in seven out of fourteen cases, and the correct algorithm (if not the optimal aspect ratio) in ten of the cases. The error in misidentifying the optimal algorithm was acceptable, especially for the "scaling" examples, $P > 8$. The performance sensitivity of choosing the wrong algorithm (but with an optimum aspect ratio) is not as extreme as when choosing the aspect ratio, but worst case errors range as high as 85%. Note that when considering a larger sampling of interesting problem sizes, all of the parallel algorithms are optimal in some cases. It is not possible to eliminate any of the parallel algorithms *a priori*.

6.4. Runtime predictions

When allocating resources, it is important to know how long a parallel job will take to run on a given number of processors. For example, runtime information is often required when submitting batch requests. This type of prediction requires a certain degree of absolute accuracy, but the degree needed is not great. (However, accurate predictions of runtime can be extremely important in real-time environments.)

Table 9 indicates how accurately the models predict the runtime for the model-determined "optimal" parallel algorithms (to pick particular examples). The percentage error is measured as in (5). With possibly one exception, the accuracy of these predictions is adequate for the determination of resource requirements. Note that similar accuracies hold for predicted speedup and parallel efficiency. The data indicate that model accuracy for problem size T42 is not scaling well beyond 256 processors, at least for algorithm TH. However, the practical limit for T42 is 512 processors, and this degradation in accuracy is not significant for this application code.

Processors	T42 algorithm	T42 predicted runtime	T42 % error in prediction	T85 algorithm	T85 predicted runtime	T85 % error in prediction
8	DR 1x8	79.8	-1.6	DT 1x8	426.6	-2.8
16	DT 1x16	40.9	-6.6	DT 1x16	206.9	-8.4
32	TR 8x4	23.0	2.2	TR 16x2	118.6	0.7
64	TR 16x4	12.2	1.7	TR 16x4	60.6	4.3
128	TH 16x8	6.7	-5.4	TT 16x8	31.6	4.5
256	TH 16x16	4.0	-11.1	TT 16x16	16.8	1.8
512	TH 16x32	2.6	-27.8	TT 16x32	9.7	-5.8

Table 9: Error in predicting runtime (seconds).

Processors	T42 model optimum	T42 experimental optimum	T42 % diff. in runtime	T85 model optimum	T85 experimental optimum	T85 % diff. in runtime
8	DT 1x8	DR 1x8	6.6	DT 1x8	DR 1x8	6.3
16	DT 1x16	DT 1x16	—	DT 1x16	DR 1x16	1.8
32	DT 2x16	TR 8x4	17.3	DT 2x16	TR 4x8	10.9
64	DT 4x16	TR 16x4	22.3	TT 16x4	TR 8x8	7.5
128	TT 16x8	TR 16x8	2.7	TT 16x8	TT 8x16	2.5
256	TT 16x16	TT 16x16	—	TT 16x16	TT 16x16	—
512	TR 16x32	TH 16x32	45.1	TT 16x32	TT 16x32	—

Table 10: Error in choosing optimal algorithm from complexity analysis instead of experimentally.

6.5. Model accuracy requirements

The previous results indicate that the accuracy of our phase model approach is adequate for algorithm tuning and comparison for this case study. We next discuss whether a simpler model might also suffice.

There are numerous ways to simplify the current model. Here we consider only a few obvious alternatives. First, we choose the optimal algorithm on the basis of arithmetic complexity alone, ignoring copy phases, communication costs, and phase-dependent rates. (Including copy and communication complexity would require some sort of rate estimation to weight the different components of the model.)

Table 10 indicates the true and predicted optimal parallel algorithms using this simplified model, and the percentage loss from using the model-identified algorithm. These predictions are not as good as those from using a phase model. Depending on the application, the size of these errors may or may not be acceptable. But, since the error in the prediction is not known in practice, the wide and unpredictable variation in the error is worrisome.

We can not predict runtimes from the complexity analysis alone. The next models we

consider use the sustained computation rate for an 8-processor run for a given parallel algorithm to weight the corresponding arithmetic complexity model. Unlike for the phase models, a separate rate was determined for each problem size. Table 11 indicates how accurately these models predict the runtime for the above model-determined "optimal" parallel algorithms. For this type of model to be accurate requires that either copy and communication costs are negligible or they scale similarly with the computation costs, and that the rates are insensitive to scaling. It is clear from Tab. 11 that these conditions do not hold for PSTSWM.

	T42			T85	
Processors	algorithm	predicted runtime	% error in prediction	algorithm	predicted runtime	% error in prediction
16	DT 1x16	41.2	-6.3	DT 1x16	205.3	-9.1
32	DT 2x16	20.8	-21.1	DT 2x16	103.3	-20.9
64	DT 4x16	10.6	-27.4	TT 16x4	50.6	-18.6
128	TT 16x8	5.5	-23.0	TT 16x8	25.7	-15.1
256	TT 16x16	3.0	-32.0	TT 16x16	13.1	-20.8
512	TR 16x32	1.6	-56.1	TT 16x32	6.9	-33.1

Table 11: Error in predicting runtime (seconds) using complexity-based model.

Our final simplified model includes terms for computation, copy, and communication costs, but does not take into account phase-specific rates. Instead we use average copy and computation rates determined from the 8-processor runs. As before, different rates are used for each parallel algorithm and problem size. Table 12 indicates how accurately this type of single-phase model predicts the runtime for the phase model "optimal" parallel algorithms (to allow direct comparison with the phase model results). With the exception of predictions for T42 for large numbers of processors, the single-phase model is as accurate a predictor of runtime as is the (multiple-) phase model. So the question arises whether a phase model is required as long as the copy, computation, and communication costs are included in the model.

	T42			T85	
Processors	algorithm	predicted runtime	% error in prediction	algorithm	predicted runtime	% error in prediction
8	DR 1x8	86.8	7.0	DT 1x8	445.4	1.5
16	DT 1x16	43.9	0.2	DT 1x16	210.9	-6.6
32	TR 8x4	23.3	3.7	TR 16x2	118.8	-0.6
64	TR 16x4	12.1	0.9	TR 16x4	60.3	3.8
128	TH 16x8	6.6	-7.2	TT 16x8	32.1	6.0
256	TH 16x16	3.8	-15.5	TT 16x16	16.9	2.2
512	TH 16x32	2.4	-32.4	TT 16x32	9.7	-5.8

Table 12: Error in predicting runtime (seconds) using single-phase model.

A phase model does not appear to be required for accurate performance prediction for

PSTSWM. However, we found the act of constructing the phase model to be necessary. The error prone aspect of the phase model approach was in the generation of the phase model expressions. These same expressions are needed in a single-phase model (or in a complexity analysis). The additional step of calculating rates and validating the individual phase models also validates the expressions. Modeling phases can also identify performance "problems", for example, code that is overly sensitive to aspect ratio due to compiler peculiarities. Using average rates and a single-phase model removes the necessity of detailed profiling to determine individual phase model rates, but makes it more difficult to validate the model.

7. Conclusions

This case study demonstrates that relatively simple algebraic models can be used to construct scalable performance models for use in algorithm tuning and comparison. These models can be difficult to generate and validate, but the phase model approach makes it feasible to do so. In addition, constructing and modifying models and generating predictions were easy using PerPreT. Note that our modeling "discipline", used to limit the amount of work spent in tuning the models, is somewhat artificial. Some restrictions are necessary for the study to be meaningful, but there may be better ways of determining phase model rates than simply running the full application for the target problem size on a small number of processors.

A phase model approach was useful in generating a performance model, but it may not be necessary when "porting" the model to a new platform. As described earlier, single rates for computation, copy, and communication phases may be sufficient when using the model for predictions. In future work, we will examine this issue by repeating our evaluation studies on the IBM SP2 and on the Cray Research T3D or T3E. The SP2 will be a particularly interesting platform; communication costs are relatively high, and a simple communication model may not be adequate.

It is clear that additional tools would be useful in generating performance models. For example, interactive tools to aid the application expert in generating the models from the source code (as in [22]), in devising experiments to determine rates and to validate models, and in calling PerPreT to make predictions would have made this process much simpler. We do not currently forsee tools that can generate performance models for complete application codes automatically, except possibly in high-level language-specific environments as proposed in [5] and [20].

This study did not address the question of how to generate the models before generating code. While our algebraic models were sufficiently accurate, a detailed complexity analysis is a requirement for an accurate comparison. Many of the costs, for example, copy phases and rates, may not be obvious until the design and implementation are fairly advanced. One

possible approach is to generate a hierarchy of models, at each step eliminating obviously bad parallel algorithms. The performance models of the remaining candidates would then be refined (possibly simultaneously generating the code). This is a big job in itself, and a sophisticated prototyping environment would be very useful. We hope that our results on the advantages and limitations of algebraic performance models will be useful in the design of such tools.

8. Acknowledgements

This research was supported by the U.S. Department of Energy under Contract DE-AC05-96OR22464 with Lockheed Martin Energy Research Inc. and by the Alexander von Humboldt foundation. The Intel XP/S 150 MP Paragon operated by the Center for Computational Science at ORNL is funded by the Department of Energy's Mathematical, Information and Computational Sciences Division of the Office of Computational and Technology Research.

9. References

[1] S. R. M. Barros and T. Kauranne, *On the parallelization of global spectral weather models*, Parallel Computing, 20 (1994), pp. 1335–1356.

[2] J. Brehm, L. Dowdy, M. Madhukar, and E. Smirni, *PerPreT - a performance prediction tool*, in Quantitative Evaluation of Computing and Communication Systems, Lecture Notes in Computer Science 977, Springer, Heidelberg, 1995.

[3] M. Calzarossa and G. Serazzi, *Workload characterization - a survey*, Proceedings of the IEEE, 81 (1993), pp. 1136–1150.

[4] D. Dent, *A modestly parallel model*, in The Dawn of Massively Parallel Processing in Meteorology, G.-R. Hoffman and D. K. Maretis, eds., Springer-Verlag, Berlin, 1990, pp. 21–31.

[5] T. Fahringer, *Estimating and optimizing performance for parallel programs*, IEEE Computer, 28 (1995), pp. 47–56.

[6] I. Foster, W. Gropp, and R. Stevens, *The parallel scalability of the spectral transform method*, Mon. Wea. Rev., 120 (1992), pp. 835–850.

[7] I. T. Foster, B. Toonen, and P. H. Worley, *Performance of parallel computers for spectral atmospheric models*, Tech. Report ORNL/TM-12986, Oak Ridge National Laboratory, Oak Ridge, TN, April 1995. (also, *J. Atm. Oceanic Tech*, accepted).

[8] I. T. FOSTER AND P. H. WORLEY, *Parallelizing the spectral transform method: A comparison of alternative parallel algorithms*, in Parallel Processing for Scientific Computing, R. F. Sincovec, D. E. Keyes, M. R. Leuze, L. R. Petzold, and D. A. Reed, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1993, pp. 100–107.

[9] ——, *Parallel algorithms for the spectral transform method*, Tech. Report ORNL/TM–12507, Oak Ridge National Laboratory, Oak Ridge, TN, May 1994. (also, *SIAM J. Sci. Comput.*, accepted).

[10] U. GÄRTEL, W. JOPPICH, AND A. SCHÜLLER, *Parallelizing the ECMWF's weather forecast program: The 2D case*, Parallel Computing, 19 (1993), pp. 1413–1426.

[11] G. A. GEIST, A. L. BEGUELIN, J. J. DONGARRA, W. JIANG, R. J. MANCHEK, AND V. S. SUNDERAM, *PVM: Parallel Virtual Machine - A Users Guide and Tutorial for Network Parallel Computing*, MIT Press, Boston, 1994.

[12] G. A. GEIST, M. T. HEATH, B. W. PEYTON, AND P. H. WORLEY, *PICL: a portable instrumented communication library, C reference manual*, Tech. Report ORNL/TM-11130, Oak Ridge National Laboratory, Oak Ridge, TN, July 1990.

[13] W. GROPP, E. LUSK, N. DOSS, AND T. SKJELLUM, *A high-performance, portable implementation of the MPI message-passing interface standard*, Tech. Report ANL/MCS-P567-0296, Argonne National Laboratory, February 1996.

[14] J. J. HACK AND R. JAKOB, *Description of a global shallow water model based on the spectral transform method*, NCAR Tech Note NCAR/TN-343+STR, National Center for Atmospheric Research, Boulder, CO, February 1992.

[15] P. HEIDELBERGER AND K. S. TRIVEDI, *Analytic queuing models for programs with internal concurrency*, IEEE Trans. Comput., c-32 (1983), pp. 73–82.

[16] T. KAURANNE AND S. R. M. BARROS, *Scalability estimates of parallel spectral atmospheric models*, in Parallel Supercomputing in Atmospheric Science: Proceedings of the Fifth ECMWF Workshop on Use of Parallel Processors in Meteorology, G.-R. Hoffman and T. Kauranne, eds., World Scientific Publishing Co. Pte. Ltd., Singapore, 1993, pp. 312–328.

[17] R. D. LOFT AND R. K. SATO, *Implementation of the NCAR CCM2 on the Connection Machine*, in Parallel Supercomputing in Atmospheric Science: Proceedings of the Fifth ECMWF Workshop on Use of Parallel Processors in Meteorology, G.-R. Hoffman and T. Kauranne, eds., World Scientific Publishing Co. Pte. Ltd., Singapore, 1993, pp. 371–393.

[18] B. MACHENHAUER, *The spectral method*, in Numerical Methods Used in Atmospheric Models, vol. II of GARP Pub. Ser. No. 17. JOC, World Meteorological Organization, Geneva, Switzerland, 1979, ch. 3, pp. 121–275.

[19] MPI COMMITTEE, *MPI: a message-passing interface standard*, Internat. J. Supercomputer Applications, 8 (1994), pp. 165–416.

[20] M. PARASHAR AND S. HARIRI, *Compile-time performance prediction of HPF/Fortran 90D*, IEEE Parallel and Distributed Technology, 4 (1996), pp. 57–73.

[21] R. B. PELZ AND W. F. STERN, *A balanced parallel algorithm for spectral global climate models*, in Parallel Processing for Scientific Computing, R. F. Sincovec, D. E. Keyes, M. R. Leuze, L. R. Petzold, and D. A. Reed, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1993, pp. 126–128.

[22] S. R. SARUKKAI, P. MEHRA, AND R. J. BLOCK, *Automated scalability analysis of message-passing parallel programs*, IEEE Parallel and Distributed Technology, 3 (1995), pp. 21–32.

[23] E. SMIRNI AND ET. AL., *Thread placement on the intel paragon: Modeling and experimenation*, in Proceedings of the 3rd International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 95), IEEE Computer Society Press, Los Alamitos, CA, January 1995, pp. 226–231.

[24] A. THOMASIAN AND P. F. BAY, *Analytic queuing network models for parallel processing of task systems*, IEEE Trans. Comput., c-35 (1986), pp. 1045–1054.

[25] H. WABNIG AND G. HARING, *PAPS - the parallel program performance prediction toolset*, in 7th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation, 1994, pp. 284–304.

[26] D. W. WALKER, P. H. WORLEY, AND J. B. DRAKE, *Parallelizing the spectral transform method. Part II*, Concurrency: Practice and Experience, 4 (1992), pp. 509–531.

[27] D. L. WILLIAMSON, J. B. DRAKE, J. J. HACK, R. JAKOB, AND P. N. SWARZTRAUBER, *A standard test set for numerical approximations to the shallow water equations on the sphere*, J. Computational Physics, 102 (1992), pp. 211–224.

[28] P. H. WORLEY, *Phase modeling of a parallel scientific code*, in Proceedings of the Scalable High Performance Computing Conference SHPCC-92, J. Saltz and R. Voigt, eds., IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 322–327.

[29] P. H. WORLEY AND J. B. DRAKE, *Parallelizing the spectral transform method*, Concurrency: Practice and Experience, 4 (1992), pp. 269–291.

[30] P. H. WORLEY AND M. T. HEATH, *Performance characterization research at Oak Ridge National Laboratory*, in Parallel Processing for Scientific Computing, J. Dongarra, P. Messina, D. C. Sorenson, and R. G. Voigt, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 431–436.

[31] P. H. WORLEY AND B. TOONEN, *A users' guide to PSTSWM*, Tech. Report ORNL/TM–12779, Oak Ridge National Laboratory, Oak Ridge, TN, July 1995.

Appendix

	DR	DH	DT	TR	TH	TT
NLLON_P	$\left\lceil \frac{\texttt{NLON}}{\texttt{PX}} \right\rceil$					
NLLAT_P	$2 \cdot \left\lceil \frac{\texttt{NLAT}}{2 \cdot \texttt{PY}} \right\rceil$		$\left\lceil \frac{\texttt{NLAT}}{\texttt{PY}} \right\rceil$	$2 \cdot \left\lceil \frac{\texttt{NLAT}}{2 \cdot \texttt{PY}} \right\rceil$		$\left\lceil \frac{\texttt{NLAT}}{\texttt{PY}} \right\rceil$
NLVER_P	NVER					
NLLON_F	$\left\lceil \frac{\texttt{NLON}}{\texttt{PX}} \right\rceil$			NLON		
NLLAT_F	$2 \cdot \left\lceil \frac{\texttt{NLAT}}{2 \cdot \texttt{PY}} \right\rceil$		$\left\lceil \frac{\texttt{NLAT}}{\texttt{PY}} \right\rceil$	$2 \cdot \left\lceil \frac{\texttt{NLAT}}{2 \cdot \texttt{PY}} \right\rceil$		$\left\lceil \frac{\texttt{NLAT}}{\texttt{PY}} \right\rceil$
NLVER_F	NVER			$\left\lceil \frac{\texttt{NVER}}{\texttt{PX}} \right\rceil$		
NLMM_S	$\left\lceil \frac{\texttt{MM}+1}{\texttt{PX}} \right\rceil$			$\texttt{MM}+1$		$\left\lceil \frac{\texttt{MM}+1}{\texttt{PY}} \right\rceil$
NLLAT_S	$2 \cdot \left\lceil \frac{\texttt{NLAT}}{2 \cdot \texttt{PY}} \right\rceil$		NLAT	$2 \cdot \left\lceil \frac{\texttt{NLAT}}{2 \cdot \texttt{PY}} \right\rceil$		NLAT
NLVER_S	NVER		$\left\lceil \frac{\texttt{NVER}}{\texttt{PY}} \right\rceil$	$\left\lceil \frac{\texttt{NVER}}{\texttt{PX}} \right\rceil$		
NLSP_S	$\left\lceil \frac{\texttt{NCSP\_S}}{\texttt{PY}} \right\rceil$		NCSP_S	$\left\lceil \frac{\texttt{NCSP\_S}}{\texttt{PY}} \right\rceil$		NCSP_S

	NCSP_S
DR, DH, DT	$(\Psi+1) \cdot \left((\texttt{MM}+1) - \frac{\texttt{PX} \cdot \Psi}{2}\right)$ where $\Psi = \left\lfloor \frac{\texttt{MM}+1}{\texttt{PX}} \right\rfloor$
TR, TH	$\frac{(\texttt{MM}+2)(\texttt{MM}+1)}{2}$
TT	$\frac{(\texttt{MM}+2)(\texttt{MM}+1)}{2 \cdot \texttt{PY}} + (6 \cdot \texttt{PX} \cdot \Phi \cdot (1-\Phi)) - 3 \cdot \Phi$ where $\Phi = \frac{\texttt{MM}+1}{2 \cdot \texttt{PY}} - \left\lfloor \frac{\texttt{MM}+1}{2 \cdot \texttt{PY}} \right\rfloor$

Table 13: Domain Decomposition Parameters

Phase	Model	Rate
		(1/a, 1/b)
	physical domain computation	
1	$12 \cdot \texttt{NLLON\_P} \cdot \texttt{NLLAT\_P} \cdot \texttt{NLVER\_P}$	4.8
	forward FFT	
2	$\lceil(\texttt{PX}-1)/\texttt{PX}\rceil \cdot 32 \cdot \texttt{NLLAT\_P} \cdot \texttt{NLVER\_P} \cdot (a + b \cdot \texttt{NLLON\_P})$	(4.5, 23.1)
3	$\lceil(\texttt{PX}-1)/\texttt{PX}\rceil \cdot 32 \cdot \texttt{NLLAT\_P} \cdot \texttt{NLVER\_F} \cdot (a \cdot \texttt{PX} + b \cdot \texttt{NLLON\_F})$	(17.7, 21.6)
5	$20 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot \texttt{NLLON\_F} \cdot (a + b \cdot \log_2(\texttt{NLLON\_F}/4))$	(3.8, 24.0)
6	$64 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F}/4)$	(4.0, 15.2)
7	$144 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F}/4)$	(10.4, 19.8)
	forward LT	
9	$(\texttt{PY}-1) \cdot 6 \cdot \texttt{NLVER\_S} \cdot \texttt{NCSP\_S}/\texttt{PY}$	4.4
10	$61 \cdot \texttt{NLVER\_S} \cdot \texttt{NLMM\_S} \cdot \texttt{NLLAT\_S}$	10.0
11	$(14 \cdot \texttt{NLLAT\_S} - 1) \cdot \texttt{NCSP\_S} \cdot \texttt{NLVER\_S}$	15.1
	spectral domain computation	
12	$13 \cdot \texttt{NLSC\_S} \cdot \texttt{NLVER\_S}$	11.5
	inverse LT	
13	$17 \cdot \texttt{NCSP\_S} \cdot \texttt{NLVER\_S}$	7.0
14	$(14 \cdot \texttt{NCSP\_S} + 10 \cdot \texttt{NLMM\_S}) \cdot \texttt{NLLAT\_S} \cdot \texttt{NLVER\_S}$	12.8
17	$40 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot (\texttt{NLLON\_F}/2 - \texttt{NLMM\_S})$	(22.1, 36.8)
	inverse FFT	
18	$70 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F}/4)$	(8.8, 20.4)
19	$40 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F}/2)$	(2.8, 18.6)
20	$(25/2) \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot \texttt{NLLON\_F} \cdot (a + b \cdot \log_2(\texttt{NLLON\_F}/4))$	(3.8, 24.0)
21	$\lceil(\texttt{PX}-1)/\texttt{PX}\rceil \cdot 20 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot \texttt{NLLON\_F}$	10.2
22	$\lceil(\texttt{PX}-1)/\texttt{PX}\rceil \cdot 20 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLLON\_P} \cdot (a \cdot \texttt{PX} + b \cdot \texttt{NLVER\_P})$	(15.2, 18.6)

Table 14: Computational models and MFlop/s or MByte/s rates for algorithms TR and TH

Phase	Model	Rate
	physical domain computation	
		(1/a, 1/b)
1	$12 \cdot \texttt{NLLON\_P} \cdot \texttt{NLLAT\_P} \cdot \texttt{NLVER\_P}$	4.8
	forward FFT	
2	$\lceil(\texttt{PX}-1)/\texttt{PX}\rceil \cdot 32 \cdot \texttt{NLLAT\_P} \cdot \texttt{NLVER\_P} \cdot (a + b \cdot \texttt{NLLON\_P})$	(4.5, 23.1)
3	$\lceil(\texttt{PX}-1)/\texttt{PX}\rceil \cdot 32 \cdot \texttt{NLLAT\_P} \cdot \texttt{NLVER\_F} \cdot (a \cdot \texttt{PX} + b \cdot \texttt{NLLON\_F})$	(17.7, 21.6)
5	$20 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot \texttt{NLLON\_F} \cdot (a + b \cdot \log_2(\texttt{NLLON\_F}/4))$	(3.8, 24.0)
6	$64 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F}/4)$	(4.0, 15.2)
7	$144 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F}/4)$	(10.4, 19.8)
	forward LT	
8	$\lceil(\texttt{PY}-1)/\texttt{PY}\rceil \cdot 32 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot \texttt{NLLON\_F}$	6.9
9	$\lceil(\texttt{PY}-1)/\texttt{PY}\rceil \cdot 64 \cdot \texttt{NLVER\_S} \cdot \texttt{NLMM\_S} \cdot (a \cdot \texttt{PY} + b \cdot \texttt{NLLAT\_S})$	(12.2, 20.6)
10	$61 \cdot \texttt{NLVER\_S} \cdot \texttt{NLMM\_S} \cdot \texttt{NLLAT\_S}$	10.0
11	$14 \cdot \texttt{NLLAT\_S} \cdot \texttt{NLVER\_S} \cdot \texttt{NCSP\_S}$	15.9
	spectral domain computation	
12	$13 \cdot \texttt{NLSC\_S} \cdot \texttt{NLVER\_S}$	11.5
	inverse LT	
13	$17 \cdot \texttt{NCSP\_S} \cdot \texttt{NLVER\_S}$	7.0
14	$(14 \cdot \texttt{NCSP\_S} + 10 \cdot \texttt{NLMM\_S}) \cdot \texttt{NLLAT\_S} \cdot \texttt{NLVER\_S}$	12.8
15	$\lceil(\texttt{PY}-1)/\texttt{PY}\rceil \cdot 40 \cdot \texttt{NLLAT\_S} \cdot \texttt{NLVER\_S} \cdot (a + b \cdot \texttt{NLMM\_S})$	(6.5, 21.9)
16	$\lceil(\texttt{PY}-1)/\texttt{PY}\rceil \cdot 40 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_S} \cdot (a \cdot \texttt{PY} + b \cdot (\texttt{MM}+1))$	(7.4, 21.6)
17	$40 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot (\texttt{NLLON\_F}/2 - \texttt{MM} - 1))$	(22.1, 36.8)
	inverse FFT	
18	$70 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F}/4)$	(8.8, 20.4)
19	$40 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F}/2)$	(2.8, 18.6)
20	$(25/2) \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot \texttt{NLLON\_F} \cdot (a + b \cdot \log_2(\texttt{NLLON\_F}/4))$	(3.8, 24.0)
21	$\lceil(\texttt{PX}-1)/\texttt{PX}\rceil \cdot 20 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot \texttt{NLLON\_F}$	10.2
22	$\lceil(\texttt{PX}-1)/\texttt{PX}\rceil \cdot 20 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLLON\_P} \cdot (a \cdot \texttt{PX} + b \cdot \texttt{NLVER\_P})$	(15.2, 18.6)

Table 15: Computational models and MFlop/s or MByte/s rates for algorithm TT

Phase	Model	Rate
	physical domain computation	
1	$12 \cdot \texttt{NLLON\_P} \cdot \texttt{NLLAT\_P} \cdot \texttt{NLVER\_P}$	4.8
	forward FFT	
2	$\lceil (\texttt{PX}-1)/\texttt{PX} \rceil \cdot 64 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F})$	(8.2, 22.3)
4	$20 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot \texttt{NLLON\_F} \cdot \log_2(\texttt{PX})$	7.5
5	$20 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot \texttt{NLLON\_F} \cdot (a + b \cdot \log_2(\texttt{NLLON\_F}/4))$	(3.8, 24.0)
6	$64 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F}/4)$	(4.0, 15.2)
7	$144 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F}/4)$	(10.4, 19.8)
	forward LT	
9	$(\texttt{PY}-1) \cdot 6 \cdot \texttt{NLVER\_S} \cdot \texttt{NCSP\_S}/\texttt{PY}$	4.4
10	$61 \cdot \texttt{NLVER\_S} \cdot \texttt{NLMM\_S} \cdot \texttt{NLLAT\_S}$	10.1
11	$(14 \cdot \texttt{NLLAT\_S} - 1) \cdot \texttt{NCSP\_S} \cdot \texttt{NLVER\_S}$	15.1
	spectral domain computation	
12	$13 \cdot \texttt{NLSC\_S} \cdot \texttt{NLVER\_S}$	11.5
	inverse LT	
13	$17 \cdot \texttt{NCSP\_S} \cdot \texttt{NLVER\_S}$	7.0
14	$(14 \cdot \texttt{NCSP\_S} + 10 \cdot \texttt{NLMM\_S}) \cdot \texttt{NLLAT\_S} \cdot \texttt{NLVER\_S}$	12.8
17	$40 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot (\texttt{NLLON\_F}/2 - \texttt{NLMM\_S})$	(22.1, 36.8)
	inverse FFT	
18	$70 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F}/4)$	(8.8, 20.4)
19	$40 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F}/2)$	(2.8, 18.6)
20	$(25/2) \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot \texttt{NLLON\_F} \cdot (a + b \cdot \log_2(\texttt{NLLON\_F}/4))$	(3.8, 24.0)
21	$\lceil (\texttt{PX}-1)/\texttt{PX} \rceil \cdot 40 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F})$	(6.0, 19.5)
23	$(25/2) \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot \texttt{NLLON\_F} \cdot \log_2(\texttt{PX})$	8.8

Table 16: Computational models and MFlop/s or MByte/s rates for algorithms DR and DH

Phase	Model	Rate
	physical domain computation	
1	$12 \cdot \texttt{NLLON\_P} \cdot \texttt{NLLAT\_P} \cdot \texttt{NLVER\_P}$	4.8
	forward FFT	
2	$\lceil (\texttt{PX}-1)/\texttt{PX} \rceil \cdot 64 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F})$	(8.2, 22.3)
4	$20 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot \texttt{NLLON\_F} \cdot \log_2(\texttt{PX})$	7.5
5	$20 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot \texttt{NLLON\_F} \cdot (a + b \cdot \log_2(\texttt{NLLON\_F}/4))$	(3.8, 24.0)
6	$64 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F}/4)$	(4.0, 15.2)
7	$144 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F}/4)$	(10.4, 19.8)
	forward LT	
8	$\lceil (\texttt{PY}-1)/\texttt{PY} \rceil \cdot 64 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLMM\_S})$	(4.2, 10.0)
9	$\lceil (\texttt{PY}-1)/\texttt{PY} \rceil \cdot 64 \cdot \texttt{NLVER\_S} \cdot \texttt{NLMM\_S} \cdot (a \cdot \texttt{PY} + b \cdot \texttt{NLLAT\_S})$	(12.2, 20.6)
10	$61 \cdot \texttt{NLVER\_S} \cdot \texttt{NLMM\_S} \cdot \texttt{NLLAT\_S}$	10.5
11	$14 \cdot \texttt{NLLAT\_S} \cdot \texttt{NLVER\_S} \cdot \texttt{NCSP\_S}$	15.1
	spectral domain computation	
12	$13 \cdot \texttt{NLSC\_S} \cdot \texttt{NLVER\_S}$	11.5
	inverse LT	
13	$17 \cdot \texttt{NCSP\_S} \cdot \texttt{NLVER\_S}$	7.0
14	$(14 \cdot \texttt{NCSP\_S} + 10 \cdot \texttt{NLMM\_S}) \cdot \texttt{NLLAT\_S} \cdot \texttt{NLVER\_S}$	12.8
15	$\lceil (\texttt{PY}-1)/\texttt{PY} \rceil \cdot 40 \cdot \texttt{NLLAT\_S} \cdot \texttt{NLVER\_S} \cdot (a + b \cdot \texttt{NLMM\_S})$	(6.5, 21.9)
16	$\lceil (\texttt{PY}-1)/\texttt{PY} \rceil \cdot 40 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLMM\_S} \cdot (a \cdot \texttt{PY} + b \cdot \texttt{NLVER\_F})$	(8.0, 21.0)
17	$40 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot (\texttt{NLLON\_F}/2 - \texttt{NLMM\_S}))$	(22.1, 36.8)
	inverse FFT	
18	$70 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F}/4)$	(8.8, 20.4)
19	$40 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F}/2)$	(2.8, 18.6)
20	$(25/2) \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot \texttt{NLLON\_F} \cdot (a + b \cdot \log_2(\texttt{NLLON\_F}/4))$	(3.8, 24.0)
21	$\lceil (\texttt{PX}-1)/\texttt{PX} \rceil \cdot 40 \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot (a + b \cdot \texttt{NLLON\_F})$	(6.0, 19.5)
23	$(25/2) \cdot \texttt{NLLAT\_F} \cdot \texttt{NLVER\_F} \cdot \texttt{NLLON\_F} \cdot \log_2(\texttt{PX})$	8.8

Table 17: Computational models and MFlop/s or MByte/s rates for algorithm DT

Aspect Ratio PX × PY	T42 Runtime (seconds)	T42 % error in model	T85 Runtime (seconds)	T85 % error in model
8x1	116.28	-2.0	543.54	1.4
4x2	103.83	-1.8	504.52	1.7
2x4	96.00	-1.1	480.68	1.5
1x8	83.46	-2.6	430.03	1.4
64x1	—	—	103.96	9.0
32x2	20.59	4.7	87.66	3.0
16x4	17.77	-1.4	77.59	2.0
8x8	16.24	-3.2	72.37	2.0
4x16	15.89	-3.6	72.26	1.6
2x32	17.72	-6.3	78.48	0.6
1x64	—	—	95.22	-5.7
128x1	—	—	—	—
64x2	—	—	53.43	7.2
32x4	10.84	3.4	45.63	1.1
16x8	9.73	-4.1	41.72	-0.2
8x16	9.14	-4.5	39.74	0.0
4x32	9.88	-7.4	42.43	-1.0
2x64	—	—	54.02	-7.9
256x1	—		—	
128x2	—		—	
64x4	—		28.88	1.6
32x8	6.17	-2.0	25.10	-4.0
16x16	5.82	-9.8	24.01	-7.4
8x32	6.26	-15.9	24.91	-8.7
4x64	—		31.28	-15.7
2x128	—		—	
1x256	—		—	

Table 18: Runtime and model error for algorithm DH.

Aspect Ratio PX × PY	T42 Runtime (seconds)	T42 % error in model	T85 Runtime (seconds)	T85 % error in model
8x1	116.14	-1.9	542.89	1.6
4x2	103.71	-1.9	495.99	3.2
2x4	95.61	-1.4	468.99	3.5
1x8	81.13	-1.7	412.94	4.1
64x1	—	—	103.99	9.0
32x2	21.22	1.4	86.86	3.8
16x4	18.78	-7.2	76.63	2.8
8x8	17.41	-10.3	72.55	0.8
4x16	17.19	-11.6	70.73	1.8
2x32	18.87	-12.3	75.69	0.7
1x64	—	—	84.04	0.4
128x1	—	—	—	—
64x2	—	—	52.67	8.5
32x4	11.93	-6.5	45.34	1.3
16x8	11.45	-18.5	41.43	-1.4
8x16	10.83	-18.5	40.81	-4.0
4x32	11.71	-19.2	42.97	-4.5
2x64	—	—	50.83	-5.7
256x1	—		—	
128x2	—		—	
64x4	—		29.60	-1.2
32x8	7.78	-21.8	26.67	-10.1
16x16	7.63	-28.2	25.81	-14.4
8x32	8.26	-29.2	26.51	-14.7
4x64	—		31.61	-16.5
2x128	—		—	
1x256	—		—	

Table 19: Runtime and model error for algorithm DR.

Aspect Ratio PX x PY	T42 Runtime (seconds)	T42 % error in model	T85 Runtime (seconds)	T85 % error in model
8x1	116.19	-2.3	544.60	0.7
4x2	111.70	-0.9	537.19	1.1
2x4	102.79	-1.8	507.60	-0.6
1x8	86.50	-5.5	438.74	-2.8
64x1	—	—	104.06	8.3
32x2	21.78	10.5	90.12	6.6
16x4	18.08	3.5	78.83	4.5
8x8	16.12	-0.2	72.20	0.9
4x16	14.67	-2.5	69.09	-4.9
2x32	19.09	-4.8	97.36	-7.7
1x64	28.26	-7.1	151.52	-8.5
128x1	—	—	—	—
64x2	—	—	54.41	13.8
32x4	11.27	8.3	44.50	7.9
16x8	9.60	-0.3	39.43	3.9
8x16	8.69	-3.4	36.66	-2.1
4x32	10.79	-4.4	50.55	-4.9
2x64	16.28	-4.7	81.43	-7.6
1x128	—	—	143.77	-8.4
256x1	—		—	
128x2	—		—	
64x4	—		28.11	10.3
32x8	6.31	0.5	23.72	1.0
16x16	5.40	-4.3	21.69	-5.9
8x32	6.80	-7.4	27.19	-3.5
4x64	9.56	-3.7	43.52	-7.3
2x128	—		78.47	-10.0
1x256	—		—	

Table 20: Runtime and model error for algorithm DT.

Aspect Ratio PX x PY	T42 Runtime (seconds)	T42 % error in model	T85 Runtime (seconds)	T85 % error in model
8x1	92.08	-4.4	481.83	-4.3
4x2	90.84	-3.5	477.54	-3.5
2x4	90.67	-2.0	481.30	-3.2
1x8	83.61	-2.8	429.89	1.5
64x1	44.07	0.6	226.37	-0.9
32x2	22.98	-0.6	113.80	1.2
16x4	12.46	-1.7	60.21	1.2
8x8	12.80	-3.3	61.38	1.5
4x16	13.81	-3.1	66.18	0.6
2x32	16.75	-6.2	76.71	-1.0
1x64	—	—	95.23	-5.7
128x1	46.91	-0.8	227.18	-0.7
64x2	23.43	0.2	112.33	1.8
32x4	12.55	-2.3	58.31	1.8
16x8	7.12	-5.4	31.84	0.5
8x16	7.42	-5.5	34.20	-1.1
4x32	8.97	-9.85	39.92	-3.7
2x64	—	—	51.89	-7.3
256x1	—		240.45	-3.6
128x2	25.83	-1.1	116.85	-0.6
64x4	13.44	-2.8	59.67	-0.3
32x8	7.54	-7.5	32.51	-3.2
16x16	4.50	-11.4	19.19	-8.5
8x32	5.18	-16.3	21.90	-10.4
4x64	—		28.77	-15.2
2x128	—		—	
1x256	—		—	

Table 21: Runtime and model error for algorithm TH.

Aspect Ratio PX x PY	T42 Runtime (seconds)	T42 % error in model	T85 Runtime (seconds)	T85 % error in model
8x1	92.10	-4.5	482.11	-4.4
4x2	89.34	-2.1	474.43	-3.0
2x4	88.81	-0.7	476.67	-2.8
1x8	80.98	-1.4	413.16	4.1
64x1	44.07	0.6	226.36	-0.9
32x2	22.29	2.2	113.91	0.9
16x4	11.99	1.7	58.10	4.3
8x8	12.16	1.2	57.81	6.6
4x16	13.40	-0.9	61.26	6.5
2x32	16.54	-5.5	70.23	4.3
1x64	—	—	83.85	0.7
128x1	46.64	-0.2	227.18	-0.6
64x2	22.79	2.8	112.51	1.5
32x4	12.14	0.5	56.64	4.2
16x8	6.96	-3.0	29.79	6.5
8x16	7.44	-4.1	31.23	6.7
4x32	9.11	-7.3	36.27	3.3
2x64	—	—	46.81	-1.1
256x1	—		240.35	-3.5
128x2	25.00	1.9	116.63	-0.6
64x4	12.85	1.1	57.38	3.0
32x8	7.20	-2.8	29.78	4.7
16x16	4.53	-7.0	16.95	3.0
8x32	5.60	-12.4	19.75	-1.1
4x64	—		25.73	-5.4
2x128	—		—	
1x256	—		—	

Table 22: Runtime and model error for algorithm TR.

Aspect Ratio PX x PY	T42 Runtime (seconds)	T42 % error in model	T85 Runtime (seconds)	T85 % error in model
8x1	92.07	-6.9	482.12	-8.6
4x2	102.03	-2.8	528.42	-5.3
2x4	100.94	-1.4	523.20	-3.6
1x8	91.17	-2.4	453.04	1.5
64x1	44.05	-2.7	228.49	-10.8
32x2	24.39	2.9	126.44	-6.5
16x4	12.95	2.5	62.14	-1.0
8x8	12.62	6.2	60.08	5.8
4x16	12.74	8.0	60.17	8.2
2x32	14.44	2.9	62.82	5.9
1x64	19.05	13.2	65.26	-1.0
128x1	46.65	-1.0	229.02	-10.4
64x2	24.80	5.5	124.70	-4.9
32x4	12.92	4.8	59.93	0.8
16x8	7.15	1.4	30.28	4.5
8x16	7.09	4.7	29.54	11.2
4x32	8.06	1.3	30.34	12.3
2x64	11.25	11.0	34.22	6.7
1x128	—	—	53.47	17.2
256x1	—		242.19	-12.7
128x2	27.38	6.1	130.04	-5.8
64x4	13.88	7.0	61.80	-0.7
32x8	7.57	2.0	30.61	3.3
16x16	4.33	1.3	16.53	1.6
8x32	5.00	-2.9	16.83	5.5
4x64	6.91	4.9	18.93	3.3
2x128	—		28.59	21.0
1x256	—		—	

Table 23: Runtime and model error for algorithm TT.

INTERNAL DISTRIBUTION

1. E. F. D'Azevedo
2. T. S. Darland
3. J. J. Dongarra
4. T. H. Dunigan
5. G. A. Geist
6. K. L. Kliewer
7–11. M. R. Leuze
12. C. E. Oliver
13–17. S. A. Raby
18–22. R. F. Sincovec
23. P. H. Worley
24. Central Research Library
25. ORNL Patent Office
26. K-25 Applied Technology Library
27. Y-12 Technical Library
28. Laboratory Records - RC
29–30. Laboratory Records Department

EXTERNAL DISTRIBUTION

31. Edward H. Barsis, Computer Science and Mathematics, P. O. Box 5800, Sandia National Laboratory, Albuquerque, NM 87185

32. Jürgen Brehm, University of Hannover, Institut für Rechnerstrukturen und Betriebssysteme, Lange Laube 3, 30159 Hannover, Germany

33. Roger W. Brockett, Wang Professor of EE and CS, Division of Applied Sciences, 29 Oxford Street, Harvard University, Cambridge, MA 02138

34. Jagdish Chandra, Army Research Office, P. O. Box 12211, Research Triangle Park, NC 27709-2211

35. Larry Dowdy, Computer Science Department, Vanderbilt University, Nashville, TN 37235

36. Geoffrey C. Fox, NPAC, 111 College Place, Syracuse University, Syracuse, NY 13244-4100

37. Dennis B. Gannon, Computer Science Department, Indiana University, Bloomington, IN 47401

38. Alan George, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

39. Gene Golub, Computer Science Department, Stanford University, Stanford, CA 94305

40. John L. Gustafson, Ames Laboratory, 236 Wilhelm Hall, Iowa State University, Ames, IA 50011-3020

41. Michael T. Heath, Department of Computer Science, 4157 Beckman Institute 405 North Mathews, Urbana, IL 61801

42. John L. Hennessy, CIS 208, Stanford University, Stanford, CA 94305

43. Dan Hitchcock, ER-31, Mathematical, Information, and Computational Sciences Division, Office of Computational and Technology Research, Office of Energy Research, U.S. Department of Energy, Washington, DC 20585

44. Charles J. Holland, Air Force Office of Scientific Research, 110 Duncan Avenue, Suite B115, Bolling Air Force Base, Washington, DC 20332-0001

45. Kenneth Kennedy, Department of Computer Science, Rice University, P.O. Box 1892, Houston, TX 77001

46. Tom Kitchens, ER-31, Mathematical, Information, and Computational Sciences Division, Office of Computational and Technology Research, Office of Energy Research, Washington, DC 20585

47. Richard Lau, Office of Naval Research, Code 111MA 800 Quincy Street, Boston Tower 1, Arlington, VA 22217-5000

48. Peter D. Lax, Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012

49. Manish Madhukar, Computer Science Department, Vanderbilt University, Box 1679, Station B, Nashville, TN 37235

50. James McGraw, Lawrence Livermore National Laboratory, L-306, P. O. Box 808, Livermore, CA 94550

51. David B. Nelson, Associate Director, Office of Computational and Technology Research, ER-30, Office of Energy Research, U.S. Department of Energy, Washington, DC 20585

52. Joseph Oliger, Computer Science Department, Stanford University, Stanford, CA 94305

53. James M. Ortega, Department of Applied Mathematics, Thornton Hall, University of Virginia, Charlottesville, VA 22901

54. Merrell Patrick, Computer and Information Science and Engineering (CISE), National Science Foundation, 4201 Wilson Blvd., Arlington, VA 22230

55. James C. T. Pool, Deputy Director, Caltech Concurrent Supercomputing Facility, California Institute of Technology, MS 158-79, Pasadena, CA 91125

56. Daniel A. Reed, Computer Science Department, University of Illinois, Urbana, IL 61801

57. Ahmed H. Sameh, Department of Computer Science, University of Minnesota, 200 Union Street S.E., Minneapolis, MN 55455

58. Rick Stevens, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439

59. Paul N. Swarztrauber, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307

60. Andrew B. White, Computing Division, Los Alamos National Laboratory, Los Alamos, NM 87545

61. Office of Assistant Manager for Energy Research and Development, U.S. Department of Energy, Oak Ridge Operations Office, P. O. Box 2001, Oak Ridge, TN 37831-8600

62–63. Office of Scientific & Technical Information, P. O. Box 62, Oak Ridge, TN 37831